LightGBM的主要优点：

1. 简单易用。提供了主流的Python\C++\R语言接口，用户可以轻松使用LightGBM建模并获得相当不错的效果。
2. 高效可扩展。在处理大规模数据集时高效迅速、高准确度，对内存等硬件资源要求不高。
3. 鲁棒性强。相较于深度学习模型不需要精细调参便能取得近似的效果。
4. **LightGBM直接支持缺失值与类别特征**，无需对数据额外进行特殊处理

LightGBM的主要缺点：

1. 相对于深度学习模型无法对时空位置建模，不能很好地捕获图像、语音、文本等高维数据。
2. 在拥有海量训练数据，并能找到合适的深度学习模型时，深度学习的精度可以遥遥领先LightGBM。

基于英雄联盟数据集的LightGBM分类实战——坏了出大事了，我不玩英雄联盟

- 我们发现不同对局中插眼数和拆眼数的取值范围存在明显差距，甚至有前十分钟插了250个眼的异常值。
- 我们发现EliteMonsters的取值相当于Deagons + Heralds。
- 我们发现TotalGold 等变量在大部分对局中差距不大。
- 我们发现两支队伍的经济差和经验差是相反数。
- 我们发现红队和蓝队拿到首次击杀的概率大概都是50%

```
## 根据上面的描述，我们可以去除一些重复变量，比如只要知道蓝队是否拿到一血，我们就知道红队有没有拿到，可以去除红队的相关冗余数据。
drop_cols = ['redFirstBlood','redKills','redDeaths'
             ,'redGoldDiff','redExperienceDiff', 'blueCSPerMin',
            'blueGoldPerMin','redCSPerMin','redGoldPerMin']
x.drop(drop_cols, axis=1, inplace=True)
```

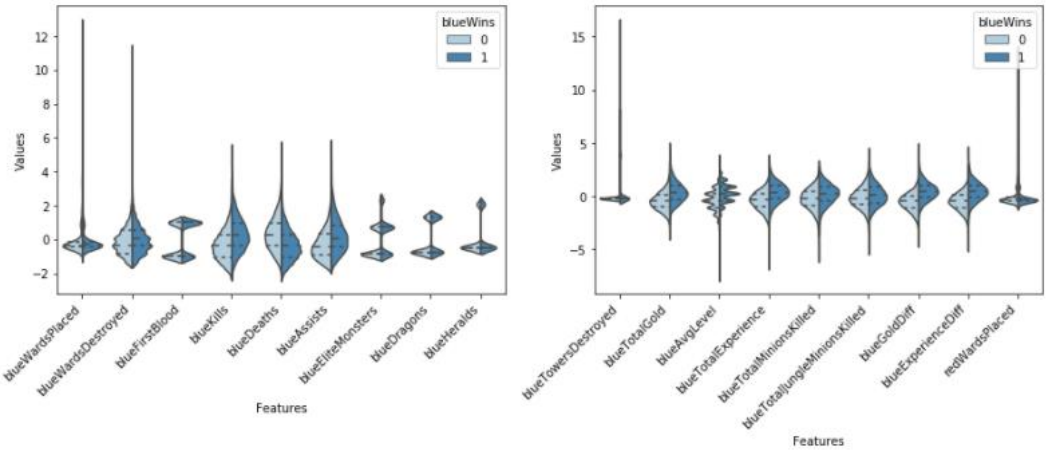
对于具体情景的分析能够减少一些错误的feature

```python
# 绘制小提琴图
sns.violinplot(x='Features', y='Values', hue='blueWins', data=data, split=True,
               inner='quart', ax=ax[0], palette='Blues')
fig.autofmt_xdate(rotation=45)

data = x
data_std = (data - data.mean()) / data.std()
data = pd.concat([y, data_std.iloc[:, 9:18]], axis=1)
data = pd.melt(data, id_vars='blueWins', var_name='Features', value_name='Values')

# 绘制小提琴图
sns.violinplot(x='Features', y='Values', hue='blueWins',
               data=data, split=True, inner='quart', ax=ax[1], palette='Blues')
fig.autofmt_xdate(rotation=45)

plt.show()
```

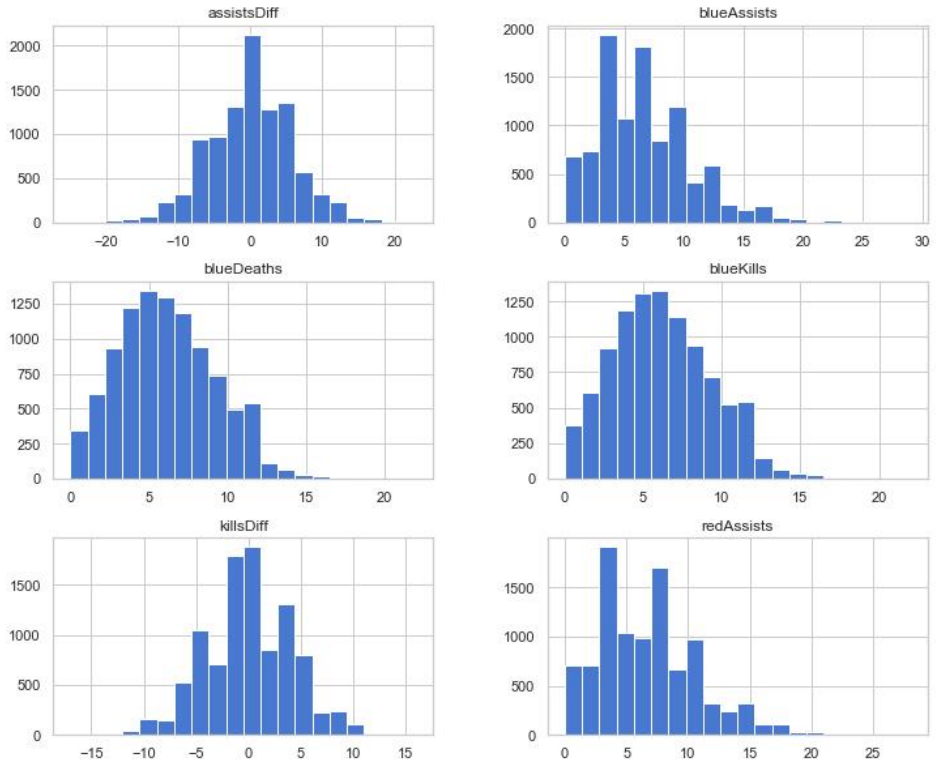


小提琴图 (Violin Plot)是用来展示多组数据的分布状态以及概率密度。这种图表结合了箱形图和密度图的特征，主要用来显示数据的分布形状。

这个小提琴图也是第一次见诶

```
x['killsDiff'] = x['blueKills'] - x['blueDeaths']
x['assistsDiff'] = x['blueAssists'] - x['redAssists']

x[['blueKills','blueDeaths','blueAssists','killsDiff','assistsDiff','redAssists']].hist(figsize=(12,10), bins=20)
plt.show()
```



我们发现击杀、死亡与助攻数的数据分布差别不大。但是击杀减去死亡、助攻减去死亡的分布与原分布差别很大，因此我们新构造这么两个特征。

LightGBM中包括但不限于下列对模型影响较大的参数：

1. learning_rate: 有时也叫作eta，系统默认值为0.3。每一步迭代的步长，很重要。太大了运行准确率不高，太小了运行速度慢。
2. num_leaves：系统默认为32。这个参数控制每棵树中最大叶子节点数量。
3. feature_fraction：系统默认值为1。我们一般设置成0.8左右。用来控制每棵随机采样的列数的占比(每一列是一个特征)。
4. max_depth： 系统默认值为6，我们常用3-10之间的数字。这个值为树的最大深度。这个值是用来控制过拟合的。max_depth越大，模型学习的更加具体。

2.4.1 LightGBM的重要参数

2.4.1.1 基本参数调整

1. num_leaves参数 这是控制树模型复杂度的主要参数，**一般的我们会使num_leaves小于（2的max_depth次方），以防止过拟合**。由于LightGBM是leaf-wise建树与XGBoost的depth-wise建树方法不同，num_leaves比depth有更大的作用。、

2. min_data_in_leaf 这是处理过拟合问题中一个非常重要的参数. 它的值取决于训练数据的样本个树和 num_leaves参数. 将其设置的较大可以避免生成一个过深的树, 但有可能导致欠拟合. **实际应用中, 对于大数据集, 设置其为几百或几千就足够了.**

3. max_depth 树的深度，depth 的概念在 leaf-wise 树中并没有多大作用, 因为并不存在一个从 leaves 到 depth 的合理映射。

2.4.1.2 针对训练速度的参数调整

1. 通过设置 bagging_fraction 和 bagging_freq 参数来使用 bagging 方法。
2. 通过设置 feature_fraction 参数来使用特征的子抽样。
3. 选择较小的 max_bin 参数。
4. 使用 save_binary 在未来的学习过程对数据加载进行加速。

2.4.1.3 针对准确率的参数调整

1. 使用**较大的 max_bin** （学习速度可能变慢）
2. 使用**较小的 learning_rate 和较大的 num_iterations**
3. 使用**较大的 num_leaves** （可能导致过拟合）
4. 使用更大的训练数据
5. 尝试 dart 模式

2.4.1.4 针对过拟合的参数调整

1. 使用**较小的 max_bin**
2. 使用**较小的 num_leaves**
3. 使用 min_data_in_leaf 和 min_sum_hessian_in_leaf
4. 通过设置 bagging_fraction 和 bagging_freq 来使用 bagging
5. 通过设置 feature_fraction 来使用特征子抽样
6. 使用更大的训练数据
7. 使用 lambda_l1, lambda_l2 和 min_gain_to_split 来使用正则
8. 尝试 max_depth 来避免生成过深的树

2.4.2 LightGBM原理粗略讲解¶

LightGBM底层实现了GBDT算法，并且添加了一系列的新特性：

1. 基于直方图算法进行优化，使数据存储更加方便、运算更快、鲁棒性强、模型更加稳定等。
2. 提出了带深度限制的 Leaf-wise 算法，抛弃了大多数GBDT工具使用的按层生长 (level-wise) 的决策树生长策略，而使用了带有深度限制的按叶子生长策略，可以降低误差，得到更好的精度。
3. 提出了单边梯度采样算法，排除大部分小梯度的样本，仅用剩下的样本计算信息增益，它是一种在减少数据量和保证精度上平衡的算法。
4. 提出了互斥特征捆绑算法，高维度的数据往往是稀疏的，这种稀疏性启发我们设计一种无损的方法来减少特征的维度。通常被捆绑的特征都是互斥的（即特征不会同时为非零值，像one-hot），这样两个特征捆绑起来就不会丢失信息。

LightGBM是基于CART树的集成模型，它的思想是串联多个决策树模型共同进行决策